Collaborative Validation of Public-Key Certificates for IoT by Distributed Caching

Minmei Wang, Chen Qian, Xin Li and Shouqian Shi
Department of Computer Science and Engineering, University of California Santa Cruz
mwang107@ucsc.edu, cqian12@ucsc.edu, xinli@ucsc.edu and sshi27@ucsc.edu

***Abstract*—Public-key certificate validation is an important building block for various security protocols for IoT devices, such as secure channel establishment, handshaking, verifying sensing data authenticity from cloud storage, and Blockchains. However, certification validation incurs non-trivial overhead on resource-constrained IoT devices, because it either requires long latency or large cache space. This work proposes to utilize the power of distributed caching and explores the feasibility of using the cache spaces on all IoT devices as a large pool to store validated certificates. We design a Collaborative Certificate Validation (CCV) protocol including a memory-efficient and fast locator for certificate holders, a trust model to evaluate the trustworthiness of devices, and a protocol suite for dynamic update and certificate revocation. Evaluation results show that CCV only uses less than 25% validation time and reduces >90% decryption operations on each device, compared to a recent method. Malicious devices that conduct dishonest validations can be detected by the network using the proposed trust model.**

I. Introduction

In recent years, Internet of Things (IoT) has attracted significant attention due to the emerging applications of industrial automation, smart devices, vehicular communication, smart cities, and smart homes [3] [22]. A widely accepted definition of IoT for the smart environment is that IoT is an interconnection of sensing and actuating devices that are capable of sharing information across platforms through a unified framework such as cloud [10]. Thus, a great amount of data, including both public and private information, will be generated, processed and transmitted by IoT devices.

Although many current IoT devices rely on a central platform to verify data authenticity. Emerging and future IoT devices, such as personal health monitors, unmanned aerial vehicles, robots, and self-driving cars, become multi-functional, self-organized, and interactive. Hence due to scalability and autonomy problems, there may not be a central platform to interconnect all these devices. Public key cryptography (PKC) enables fundamental security protocols for IoT data communication, based on a well-functioning public key infrastructure (PKI). We list the following (incomplete) important use cases of PKC for IoT. **1)** The authentication process in protocols for establishing a secure channel between two end devices or one device and a server. For example, in an IoT-based healthcare system, wearable sensors that collect human-related data need to securely communicate with other sensors, caregivers and doctors [20]. Existing approaches modify traditional end-to-end IP security protocols to adapt to IoT environments, such as DTLS [23] and HIP DEX [21], which rely on PKC for handshaking. **2)** When an IoT device retrieves sensing data that were collected by other sensors and stored in the cloud, it needs to verify the data integrity and authenticity to guarantee that data have not been tampered with or partially dropped [18]. Digital signatures of sensing data are applied to this situation. In order to verify the correctness of a data signed by the private key of the data generator, a device first needs to validate the public key via its certificate. **3)** Recently studied Blockchain-based IoT systems [6] heavily rely on PKC. **For all situations, certificate validation is an essential step.** Although certificate validation can be completed relatively easily on an ordinary computer, it incurs non-trivial overhead on resource-constraint IoT devices. For example, using an optimized method that requires only one signature verification, certificate validation still costs 1.9 seconds and certificate-based public key operations demand 95% of the overall processing time of handshaking on the WisMote platform [19], as reported in [11].

This work focuses on a specific yet important problem: how to perform **fast certificate validation** in a large IoT network. We do **not** intend to improve handshaking protocols, PKI, or PKC schemes in IoT. Instead, we study the certificate validation method that is compatible with most existing PKIs and PKC algorithms. There have been existing work on reducing certificate validation cost in classic network environments. One method is to delegate certificate validation to a third party [11] [20],which creates a single point of failure. Prefetching and prevalidation are also used for efficient certificate validation [24], but they require heavy storage cost.

Fast certificate validation on IoT devices seems to be a dilemma: the most effective approach is to cache as many validated certificates as possible, but it is not allowed on IoT devices with limited memory. We call the process of validating a public-key certificate via verification of CA signature as *individual validation*. Individual validation of every certificate is time-consuming. Hence none of the above methods is desired for IoT.

In this work, we propose to utilize the power of distributed caching and explore the feasibility of using the cache spaces on all IoT devices as a large pool to store validated certificates, which can be accessed by any internal device. We design a **C**ollaborative **C**ertificate **V**alidation protocol (CCV), which adopts the cooperation strategy in a large IoT network and utilizes the overall computation power and storage resources. When one device d needs to validate a certificate that has been validated and cached by another device h in the network, d can request a collaborative certificate validation from h to confirm that the requested certificate matches the cached one. The design of CCV includes **three main challenges**. First, how each device can efficiently locate the holder of a certificate without storing a long index that maps every certificate to its holder. Second, how to avoid false validation results shared by the IoT devices controlled by the attacker (called malicious devices). Third, how to dynamically maintain a consistent collaborative validation when new certificates are validated and cached certificates are removed or revoked.

Our contributions of this work include the following. 1) We design a memory-efficient and fast locator for certificate holders, called OLoc, based on a recent data structure Othello Hashing [26]. 2) We introduce a trust model for CCV to evaluate the trustworthiness of each device to avoid dishonest collaborative validation from malicious devices. 3) We design a complete protocol suite for efficient OLoc update, cache replacement, and revocation status checking mechanisms in a dynamic network. Evaluation results show that CCV only uses less than 25% time compared to the certificate validation in a recent method [11]. The majority time cost of CCV is on network latency rather than local public key decryptions (reducing $> 90\%$ decryptions), hence it significantly saves computation resource.

The paper is structured as follows. We give the problem statement, network model and security model in Section II. Section III presents the design consideration of the certificate locator. We present the detailed protocol design in Section IV. We show the evaluation results and security analysis in Section V. Section VI presents the related work and Section VII concludes this work.

II. Problem Statement and Models

A. Problem Specification and Network Model

We consider a large IoT system including a large number (100 or more) of devices. The use cases of such system can be an industrial IoT network [15], a community network with home IoT devices, or an organization/building/campus network with various devices (cameras, sensors, smart office products, etc.). The system consists of the following units.

(1) IoT devices. An IoT device (or "device" in short) is a sensor or actuator with constrained computing, memory, and power resources. Each device can communicate to the Internet through the routers in the IoT system. A device sends and receives packets to/from a router using its wireless chip via either a direct connection to a router ("infrastructure mode", such as those in a home WiFi network) or multi-hop forwarding ("ad-hoc mode", such as those in a low-power sensor network). Devices can communicate with each other.

(2) Routers. Routers are the forwarding units to support communication among IoT devices, or between a device and the Internet.

(3) Tracker. A tracker is a function running on a remote or edge server to help to manage the IoT network. All IoT devices can communicate with the tracker. There could be multiple duplicate trackers in a network, running on different servers. Every tracker maintains the same network state and does **not** actually perform validation or caching for devices. **A tracker in CCV is not a single point of failure.** Our experiments will show that a tracker requires minimal computation and communication cost, hence it can be easily replicated. Note that trackers do not need strong consistency or synchronization. Even if some of the trackers have inconsistent, incomplete, incorrect information, or stop functioning for a duration of time, IoT devices can still perform correct certificate validation –**trackers impact on certificate caching efficiency rather than verification correctness**. Replicated trackers using existing protocols to tolerate Byzantine failures [4], which is out of our scope.

This work focuses on the public key certificate validation problem of an IoT system. Each certificate is *uniquely identified* by its public key. We assume secure communication channels have already been established among the IoT devices and the tracker in the same network using standard IoT security solutions such as that in WirelessHART [15]. Hence a device does *not* need to validate public keys of other devices and the tracker in the same network. A device needs to validate a public key certificate of an external node from the Internet in the following situations:

(1) Authenticate an external node during the handshaking to establish a secure session, such as that in DTLS [23] [11].

(2) Verify the authenticity of the data retrieved from a cloud, which carry the digital signatures of external nodes [18].

The *collaborative certificate validation scheme* investigated in the paper can be modeled as follows. When a device receives a public key certificate that has already been verified and cached by another device in the network (called the holder), the device needs to locate the cached certificate and ask the holder to confirm it. Otherwise, it needs to run individual validation. Each device caches a (limited) number of certificates validated by itself. When a device receives a request from another device in the network to validate a certificate, it will respond based on the result from its cache. This research includes **three main challenges**. First, how each device can efficiently locate the holder of a certificate without storing a long index that maps every certificate to its holder. Second, how to void false validation results shared by the IoT devices controlled by the attacker (called malicious devices). Third, how to dynamically maintain a consistent collaborative validation when new certificates are validated and cached certificates are removed or revoked.

B. Security Model

We assume the internal communication among IoT devices or between a device and the tracker is secure. The secure communication channels have been established using standard solutions such as the security protocol of WirelessHART [15]. Group keys and session keys have been successfully distributed. We do not consider attacks on the communications between two IoT devices or a device and the tracker. All devices, except those controlled by an attacker, are willing to collaborate. The goal of a device is to maximize the functioning of the entire network rather than maximizing the functioning or lifetime of itself.

This work focuses on the research problem of efficient validation of public key certificates, assuming there exists a well-functioning PKI. This research is not about building a better PKI. Hence we do not consider attacks during the PKI validation process.

An attacker can control a number of devices in the network to conduct malicious behaviors, which are referred as *malicious devices*. Malicious devices are "malicious-but-cautious" and may collude. The tracker stores a *trust value* for every device to indicate the likelihood that the device is legitimate. We list six major attacks from malicious devices.

(1) **False validation attack:** A malicious device provides false certificate validation results to other devices.

(2) **Self-promoting attack:** A malicious device promotes its trust value by claiming that it helped other devices validate certificates. However, it did not.

(3) **Defamation attack:** A malicious device claims that a legitimate device provides wrong certificate validation results.

(4) **Traitor attack:** When a diplomatic attacker senses their reputation is dropping because of providing malicious devices, it can provide good services for a period of time to gain a high reputation. Then it provides malicious services after it gains high reputation.

(5) **Whitewashing attack:** Attackers can discard their current identities and re-enter the systems when they have very low trust levels and cannot be selected as collaborators.

(6) **Collusion attack:** Two or more malicious devices improve their trust values by claiming that they helped each other, However, they provide false validation results when helping other devices to validate certificates.

In addition, a device never individually validates a certificate that is not required by its own need, in order to avoid DoS or resource exhaustion attacks. It only caches a certificate validated by itself in prior communications and provides validation confirmation of this certificate to another device.

III. Design Consideration of Certificate Locator

One major challenge of collaborative certificate validation is to allow each device to efficiently locate another device that validated and caches the certificate of the public key to use. A simple solution is to let each device maintain a complete index of all certificate-to-device mappings. This method is not scalable because every mapping requires more than 1000 bits of memory, assuming a public key is 1024-bit long. A more advanced method is that each device maintains $m-1$ counting Bloom filters (CBFs) [7] (m is the number of devices in the network). Each CBF represents the set of certificates of a device. The drawbacks of this method are 1) locating the holder of a certificate requires up to $m-1$ Bloom filter lookup operations, and 2) CBFs are not memory-efficient. Hence these methods are impractical for IoT.

In this work we utilize and improve a recent innovation called Othello Hashing [26] to design a memory-efficient and fast locator for certificate holders. In addition, existing design of Othello Hashing does not fully satisfy the requirement of the locator hence we propose an improvement design called Othello-based Locator (OLoc). Every device stores an OLoc.

Othello Hashing is used to represent a set of key-value pairs. Given a set of keys K and each key k is mapped to a value $v \in V$. Let $n = |K|$. An Othello Hashing structure is a seven-tuple $< m_a, m_b, h_a, h_b, \mathbf{a}, \mathbf{b}, G >$ defined as follows: Integers m_a and m_b is the size of Othello. $m_a = m_b \approx 1.33n$. A pair of uniform random hash functions $< h_a, h_b >$ maps keys to integer values $0, 1, ..., m_a - 1$ and $0, 1, ..., m_b - 1$ respectively. **a** and **b** are two arrays including m_a and m_b elements respectively. G is a bipartite graph which is used to determine the values in **a** and **b**.

Othello uses $O(n)$ time to build a bipartite graph G, which is used to assign the elements of **a** and **b**, such that $\mathbf{a}(h_a(k)) \oplus \mathbf{b}(h_b(k))$ is the value of k. Hence finding the value of a given key k is extremely fast. Othello can simply retrieve $\mathbf{a}(h_a(k))$ and $\mathbf{b}(h_b(k))$ and compute their XOR, requiring only two memory access operations. In addition, G is needed only by construction, hence the devices that perform lookups (such as the IoT devices) do not need to maintain G.

Complexity. The space cost of the two arrays in Othello is small, around $2.66nl$ bits, where l is the length of each value. Each lookup only requires two memory access and one XOR operations –very small constant. Space and lookup cost is more efficient than most existing main-stream hash tables including Cuckoo Hashing [26]. The expected time to add, delete and update a key-value pair is proved to be $O(1)$ [26].

Opportunities and challenges of using Othello. We find that Othello Hashing is a good fit for the application of memory-efficient certificate locator. Let each key be a public key and the corresponding value be the holder of the certificate. We realize that, to perform locator lookups, only the two hash functions $< h_a, h_b >$ and arrays **a** and **b** need to be stored in a device. The construction information, such as the key-value pairs and the bipartite graph G are shared by the entire network and not needed for lookups. Hence this information can be stored at the tracker. Note the Othello construction and update operations are relatively more complex than lookups. Hence the IoT devices can avoid these operations and only be responsible for efficient lookups. The tracker has plenty of resources for construction and is responsible for updating and sending the updated Othello arrays to the devices.

However, one limitation of Othello Hashing is that, if we search a key k' that is not in K, Othello will return an arbitrary value. It is because $\mathbf{a}(h_a(k'))$ and $\mathbf{b}(h_b(k'))$ will be two arbitrary elements. Hence if a certificate C is not cached by any device in the network, the locator will point to an arbitrary device. Falsely locating a holder will waste both communication bandwidth and latency. In the next section, we will present an improved design of an *Othello-based Locator* (OLoc) to reduce the rate of false holder locating.

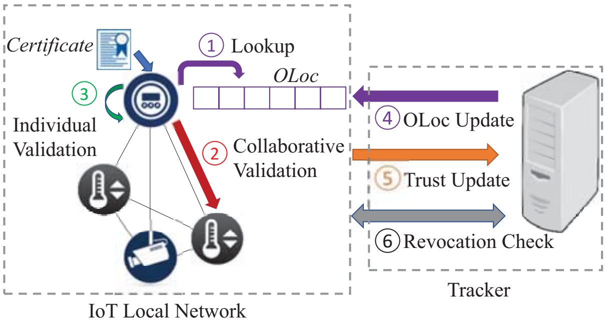


Fig. 1. Protocol overview of CCV

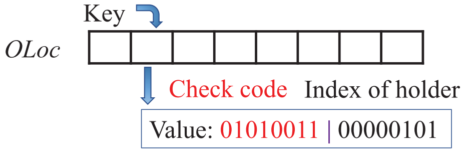


Fig. 2. OLoc lookup in CCV

IV. Protocol Design

A. Protocol Overview

Fig. 1 illustrates the overview of the proposed protocol CCV. The CCV protocol runs on both the IoT devices and the tracker.

Protocol on an IoT device d**.** When a device d needs to validate a certificate C, CCV works as follows. **Part 1.** d searches the Othello-based Locator (OLoc) to look for a holder of C. **Part 2.** If the OLoc indicates that there is a holder h of C, devices d and h conduct collaborative validation based on the cached certificate on h. **Part 3.** If the OLoc indicates that there is no holder of C, d runs individual validation. **Part 5.** If a holder h confirms the validation of C, d will forward this event to the tracker with a probability to allow the tracker to monitor the trustworthiness of h.

Protocol on the tracker. In fact, it can be any of the trackers. For simplicity, we use 'the' tracker. The tracker is responsible for updating the OLoc of different devices, monitoring the trust values, and removing revoked certificates. The protocol on the tracker operates as follows: **Part 4.** Since the certificates cached in the network change gradually, the tracker needs to update the OLoc for each device to keep track of the update-to-date holder information. It then distributes the updated OLoc to each device. **Part 5.** When the tracker receives a forwarded validation confirmation showing that h just helped d, it will verify the correctness of this confirmation and update h's trust value accordingly. **Part 6.** When the tracker receives new revocation lists from CAs, it notifies the holders to remove these certificates from their caches.

B. Othello-based Locator and Update

The CCV protocol is driven by messages and events. Upon receiving a validation requirement of a certificate C of a public key k^+ from the upper layer, a device d first checks its local cache to see whether it has cached C. If C is cached and the two versions are identical, then d confirms the validity of C. Otherwise, d needs to determine whether there is another device being the holder of C and which device it is.

Every device stores an Othello-based Locator (OLoc). The lookup key of OLoc is a public key (identifier of a certificate) and the lookup result should indicate the holder of the certificate. As discussed in Sec. III, one limitation of Othello Hashing is that, if a certificate C is not cached by any device in the network, the locator will point to an arbitrary device. Assume the network has n devices and each device can be referred by a l-bit index: $l = \lceil \log_2 n \rceil$. Our innovation is to extend the lookup value τ of an Othello to $l + l'$ bits for the certificate C of the public key k^+. The l least significant bits (LSBs) of τ is the index of the holder i and the l' most significant bits is the check code c of this certificate. c is determined by the hash value $H(k^+)$ using a CRC hash function H. Since $H(k^+)$ is longer than l' bits, c can simply be the l' LSBs of $H(k^+)$.

Fig. 2 illustrates an example of the OLoc lookup. When a device searches its OLoc for the holder of the certificate C of k^+, it compares whether the l' LSBs of $H(k^+)$ matches the check code c return by OLoc. If they match, it is highly likely that the certificate is actually cached by the holder. By "highly likely", we mean that there is still a probability that C is not cached but matches the check code, called a *false matching*. Such probability is around $1/2^{l'}$ depending on the length of the check code l'. The existence of false matchings does not hurt the correctness and security of CCV, but will slightly increase the communication cost. In the example of Fig. 2, both l and l' are set to 8, which can be adjusted based on the system requirements.

When the device d gets index i and the check code matches $H(k^+)$, d sends message $< \mathit{REQUEST\_VALI}, C, d, h >$ to another device h whose index is i to perform collaborative validation. If the check code does not match $H(k^+)$, the device terminates CCV and conducts individual validation.

On the tracker side, the OLoc at every device should be dynamically updated to reflect the update-to-date certificate to holder mapping. At the very beginning, Othello is empty. Then the tracker updates the OLoc of all devices at a fixed interval and distributes newly updated OLoc to the devices. Although the tracker is responsible for updating all devices in the network, *these updates are efficient and scalable because all devices may share a same OLoc*. When a device caches a certificate, it sends a *NEW_CACHE* message to notify the tracker. Hence the tracker keeps track of all cached certificates in the network and updates the OLoc. The updated OLoc is then sent to the devices using the *UPDATE_OLOC* message.

We apply another optimization based on the IoT network features. It is possible that one certificate C is cached by multiple devices in the network. Hence C may have multiple holders, any of which can be a valid result of a holder locator. In the construction stage of Othello, we choose the index of one holder to be the lookup result of OLoc for the public key k^+ in C. However, it is reasonable to choose the most suitable holder of C for different devices when there are multiple feasible options. To construct the OLoc of a device d, CCV may choose the holder with shortest network distance (e.g., smallest hop count) to d. One may note that using this optimization, the OLoc in different devices may be different. Two devices on different locations in the network may be close to different holders. However, constructing these different versions of OLoc is still efficient. They share the same set of keys and the same bipartite graph G, because G depends only on the set of keys, rather than their values. Note that computing G is the most time-complex step during the construction of an Othello. Once G is obtained, determining the arrays **a** and **b** is trivial. Hence all devices can still share a same G and the arrays **a** and **b** can be computed in a short time. In addition, many devices in network proximity are still able to use a same OLoc.

In addition, the trust values of IoT devices maintained by the track are used to filter malicious devices. Hence the tracker will only select the holders whose values are above a predetermined threshold.

C. Collaborative Validation

To request a collaborative validation of certificate C, device d sends a message $<$ *REQUEST_VALI*$, C, d, h >$ to the holder h. Upon receiving this message, the holder h searches its cache to find the certificate of the public key k^+ on C. If such certificate exists and is identical to C, it replies d with a message $<$ *REPLY_VALI*$, C, h, d, `Correct' >$. If the certificate exists but is different from $C$, it replies $d$ with a message $<$ *REPLY_VALI*$, C, h, d, `Wrong' >$. If no certificate of k^+ is cached, it replies d with a message $<$ *REPLY_NO_CERT*$, C, h, d >$. The main reason for a missing certificate is that previously cached certificates may be replaced by others due to the lack of cache space, while this information has not been updated to OLoc. The other reason is the false matchings. Note all these messages should be signed by h's private key for non-repudiation purposes.

Once the device d receives the *REPLY_VALI* message with a 'Correct' value from a holder h, it knows the validation of certificate C is confirmed and it can use C for incoming communications. If d receives the *REPLY_VALI* message with a 'Wrong' value. It will discard the certificate C and still forward this event to the tracker. If d receives the *REPLY_NO_CERT* message, it runs individual validation.

One additional step is that d may forward the confirmation events to the tracker, in order to improve the trust value of the holders that provide validation. It sends a message $<$ *UPDATE_TRUST*$, d, t, E >$, where E includes one or more *REPLY_VALI* messages received during the past period of time as well as their digital signatures. In order to save message cost and reduce tracker overhead, d does *not* forward every collaborative validation event but on a sampling basis. When a malicious device keeps providing false validation results, then eventually it will be detected.

D. Individual validation and caching

If the device d chooses to run individual validation of certificate C, this process consumes computation resource and relatively long latency on d. After C being validated, d will cache C in its local memory. One of the following three cache replacement strategies will be used to replace old certificate: random, FIFO (first in, first out), and LRU (Least recently used). If an old certificate C_o is replaced by a new one, d sends a message $<$ *DELETE*$, C_o, u, t >$ to the tracker.

Besides validating the certificate, d also needs to check the revocation status of the certificate. It sends $<$ *CHECK_REVO*$, C, d, t >$ to the tracker to query the revocation status. If the certificate is included in the revocation list stored in the tracker. The tracker will send a $<$ *REVO_CERT*$, C, d, t >$ message to call back C and let d stop using C and remove it from the cache.

E. Trust Model and Updates

1) Trust Model: The trust model is used to facilitate the detection of malicious devices and make them in lower probability to be the selected holder. CCV adopts the following definition of trust built from an existing model [5].

Definition 1. In the IoT network, a device d's trust to another device d' is the subjective expectation of d of receiving positive outcomes through the communications with d'.

Specifically, the trust value in CCV quantifies the expectation to receive correct validation results of certificates. The range of trust value between two devices is $[0, 1]$. At the beginning, the trust value between every two devices is set to be 0.5. Trust can be categorized into two classes: direct trust and indirect trust. **Direct trust** is the trust that is calculated by direct communications between two devices. **Indirect trust** is the trust that is calculated by indirect recommendations, which will be explained later.

We consider both direct and indirect trusts and use past communications between two devices to measure the trust value. Direct trust is based on the certificate validation results between a validation requester and the holder. During a period of time, the tracker will gather collaborative validation events and verify whether the holder honestly validated the certificates or not. At time ρ, the tracker records the number of honest validation events s and the number of dishonest validation events f between a requester d and a holder d' from the beginning of the system. We adopt a subjective logic framework [14] [13] to compute the direct trust. Due to space limit, we skip the detailed formulas.

Indirect trust is based on the recommendation. Device A trusts B with a direct trust value α and device A trusts C with a direct trust value ϵ. When the trust value that B trusts C updates to β, if the trust value from A to B exceeds the threshold θ, θ is set to be 0.5. B can recommend C to A. A trusts C with an updated value $\alpha\beta + (1 - \alpha)\epsilon$.

2) Trust Updates: When the tracker receives an *UPDATE_TRUST* message, it verifies the collaborative validation events in E. The tracker maintains two arrays A and B to store the numbers of honest and dishonest collaborative validation events between two devices respectively. $A[i][j]$ denotes the number of honest validations that i provides to j, and $B[i][j]$ denotes the number of dishonest validations that i provides to j. The tracker also stores the trust value $T[i][j]$ at current time which denotes the degree that i trusts j. Once the tracker verifies the collaborative validation event that j provided to i, it updates $A[i][j]$ or $B[i][j]$.

If the tracker finds that a validation is dishonest and the certificate is not valid, it sends a *FALSE_VALI_RESULTS* message to tell the device.

F. Revocation Check

Revocation status check is important in certificate validation but time-consuming on devices. In CCV, the tracker actively downloads the Certificate Revocation List (CRL) [17]. A device can send a *CHECK_REVO* message to request the revocation status of a certificate. This design makes the device start using the certificate simultaneously while waiting for the reply from the tracker. The tracker replies about the status using a *REPLY_REVO* message. If the certificate is revoked according to the tracker, the device stops using it, removes it from the cache, and rolls back to the prior state. In addition, when the tracker updates its local revocation list and finds existing certificates cached in the network are expired, it will send *REVO_CERT* messages to the holders of these certificates for removing them.

V. Experimental Results and Security Analysis

We implement a complete version of CCV in a packet-level discrete-event simulator running on a desktop with 3.6GHz Intel(R) Core(TM) i7-7700 CPU. The actual processes of cryptographic operations are implemented and the latencies, including cryptographic latency and network latency, are simulated. The reason is that the cryptographic latency on a desktop does not reflect the actual cryptographic overhead on an IoT device. We use the latency data of cryptographic operations gathered from a WisMote platform featuring a 16MHz MSP430 micro-controller [19] [11]. We use SHA256 for hash operations, elliptic curve NIST P-256 for PKC, and AES-128 for symmetric-key operations in secure communications among in-network IoT devices. We compare CCV to an advanced method of individual certificate validation [11], in which validating the certificate chain only requires one single decryption operation. The average time of such individual certificate validation on WisMote is around 1.9sec with 13.9ms standard deviation as reported in [11]. Note most existing certificate validation methods typically require multiple intermediate certificates in a chain, and the validation overhead grows linearly with the number of intermediate certificates [11].

In our experiments, we simulate a number of IoT devices running the CCV protocol to collaboratively validate the certificates. We use 'I-Valid' to refer to individual validation. For a fair comparison, in CCV we assume at the system start time, no certificate is validated and cached in the network. Also, OLoc will use the cache space. Every certificate in CCV must be individually validated once and cached for further use. The tracker updates the OLoc every five seconds.

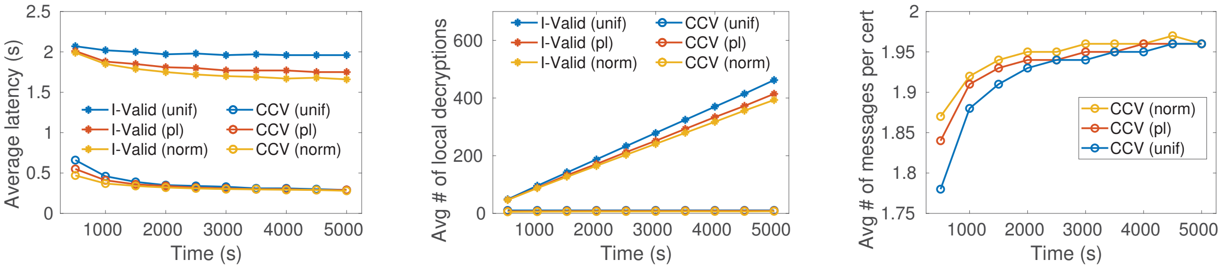
We evaluate and compare the following six metrics of CCV. 1) **Latency** is the average time from receiving a certificate to finishing validation on a device. 2) **Number of local decryptions** is the number of times of running public key decryption to validate certificates. It characterizes the computation overhead on devices. 3) **Average number of messages per certificate** evaluates the communication cost. 4) **Throughput** is the maximum number of certificate validations on the simulator. Although the computation resource on the simulator is different from that on a device, this metric still reflects whether CCV reduces resource overhead. 5) **Computational cost for OLoc update** measures the overhead of the tracker. 6) **Trust value changes** are used to detect malicious devices.

The number of events that require certificate validations happening on devices follows the Poisson distribution. The parameter λ denotes the average number of events happening in 1 sec, which is used to adjust the frequency of events. We vary the number of events that requires a particular certificate in three distribution: uniform, normal, and power law.

A. Evaluation Results

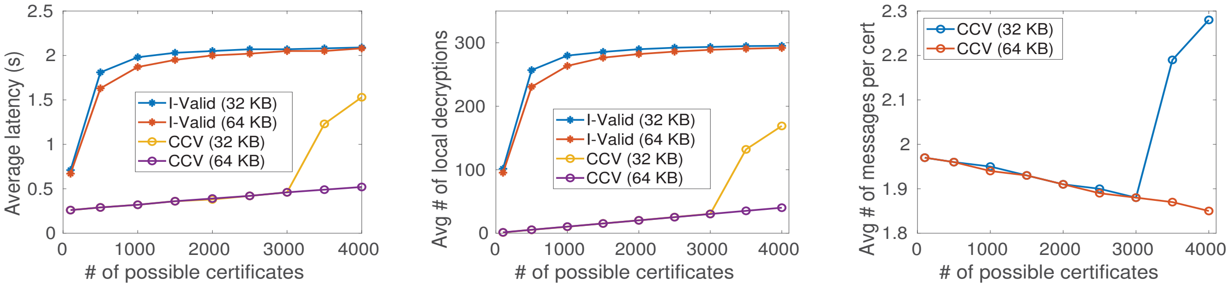
Performance varies with time. Assuming at the system start time, no certificate is validated and cached in the network. Then the devices validate and cache certificates gradually. We may expect that the validation latency of CCV will decrease when time increases. Fig. 3 shows the performance comparison of CCV and I-Valid, by varying the time. In this set of experiments, the number of IoT devices is 100, the total number of certificates is 1000, λ is set to be 10, and the memory size (OLoc and cache) is set to be 32 KB. I-Valid also uses cache space to store certificates. We show the results for uniform, normal, and power law distributions. From Fig. 3(a), we find that CCV requires > 0.5s in average to validate a certificate at time 500s and the latency keeps decreasing to 0.27s at time 5000s. CCV only uses 25% time compared to I-Valid. More importantly, Fig. 3(b) shows that CCV always requires around 10 decryptions per device, while this number can be > 400 for I-Valid at time 5000s. CCV reduces more than 99% local decryption operations and significantly saves the computation cost. The latency of CCV is mainly the network latency. From Fig. 3(c), we can see that the average number of messages for each certificate in CCV increases during time 0s to 2000s, but will be stable after 2000s. It is because more collaborative validation will be used than individual validation. CCV is very communication-efficient: the number of messages per certificate is close to two. We also find that different distributions have a relatively small influence on the performance of CCV.

Varying the number of certificates and cache size. We conduct experiments by varying both the number of total certificates and the cache size to evaluate their influence. In this set of experiments, the number of devices is 100, λ is set to be 10, and the time is a 3000s duration. The results are shown in Fig. 4. We find that CCV outperforms I-Valid protocol



(a) Validation latency (b) Number of local decryptions (c) Average # messages per certificate

Fig. 3. Performance varies with time



(a) Validation latency (b) Number of local decryptions (c) Average # messages per certificate

Fig. 4. Performance varies with the number of overall certificates and cache size (32KB and 64KB)

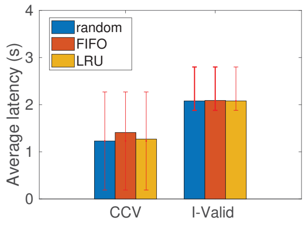


Fig. 5. Validation latency with cache replacement strategies

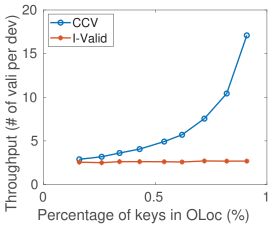


Fig. 6. Throughput comparison on simulator

with lower latency and lower number of local decryptions. When the number of total certificates increases, the latency (Fig. 4(a)) and the number of decryptions (Fig. 4(b)) both increase slightly in CCV, but still provides huge advantages compared to I-Valid. There is a sudden increase at 3000 certificates in both latency, the number of decryptions, and message cost (Fig. 4(c)) for CCV when the cache is 32KB. This is because the more cache misses are caused by increasing number of certificates. When the cache is 64KB, there is no such problem.

Cache replacement. This set of experiments vary cache replacement strategies including random, FIFO and LRU. Fig. 5 shows that the strategy has little influence on the latency of CCV, with random and LRU being slightly better.

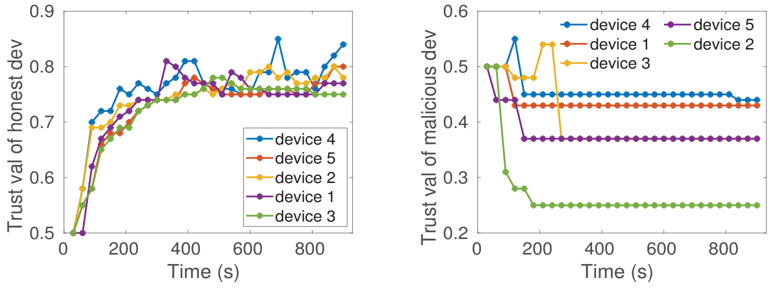
Throughput. In this set of experiments, we compare the validation capacity of CCV and I-Valid by keeping devices performing validations in the simulation. The simulator simulates 100 devices simultaneously. The signing and verification algorithm is ECDSA with 160 bits keys. Fig. 6 shows that the number of validations on a device in one millisecond. From Fig. 6, we can see that CCV has a much better throughput compared to I-Valid protocol, especially when large percentage of public keys are recorded in OLoc.

B. *Tracker overhead*

The heaviest task for the tracker is to update the OLoc. Table. I shows the time to construct an OLoc with different number of certificates. The results show that it is very time-efficient for the tracker to update the OLoc with a gigantic size of certificates. CCV also builds different OLoc to choose the most suitable holder of C for different devices when there are multiple choices. The reasons for multiple copies of C are the update delay of OLoc at the begining and the detection of malicious devices. The number of different values among different OLoc is small. Table. II shows the time to construct an OLoc based on an existing OLoc with 10000 shared keys but different values. The results show that it is very time-efficient for the tracker to build the OLoc when there is little difference in values with an existing OLoc.

C. *Security Results and Analysis*

We provide the evaluation results and analysis of how CCV defends against three major attacks conducted by malicious devices mentioned in Sec. II-B.



(a) Trust values of honest devices

(b) Trust values of malicious devices

Fig. 7. Trust values

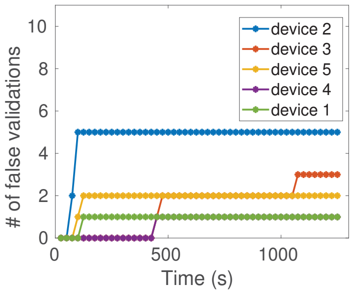
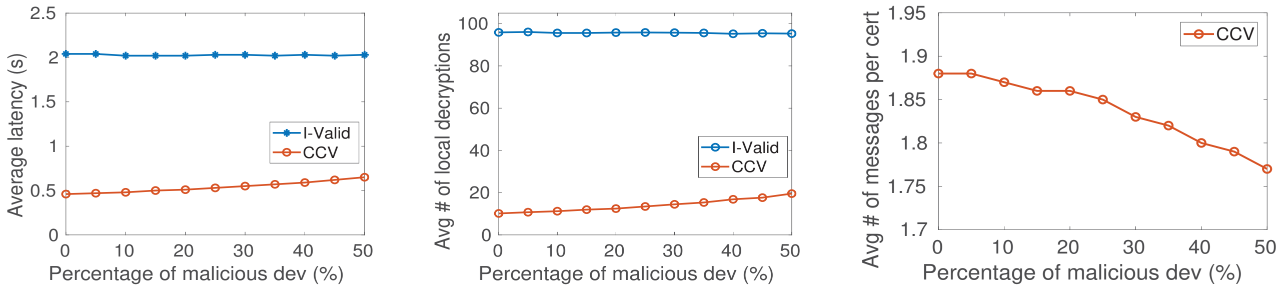


Fig. 8. Number of false validations



(a) Validation latency

(b) Number of local decryptions

(c) Average # messages per certificate

Fig. 9. Performance varies with percentage of malicious devices after 1000s

TABLE I
TIME FOR BUILDING OLOC

# certificates	100	1000	5000	10000	100000
Times (ms)	0.20	1.48	6.59	13.03	131.51

TABLE II
TIME FOR BUILDING OLOC WITH SAME KEYS BUT DIFFERNET VALS

# different vals	10	100	200	300	400
Times (ms)	2.05	2.98	4.33	5.98	7.52

Trust value evaluation. We monitor the trust values changes for both honest and malicious devices. In the set of experiments, the number of devices is 100, the number of certificates is 1000, time is set to a 900s duration, and the cache size is 32 KB for each device. The initial trust value between any two devices is set to be 0.5. The trust value will be dynamically updated due to collaborative validation. The percentage of malicious devices is 5%. Fig. 7 shows the trust value changes. We show the results of five randomly chosen honest (Fig. 7(a)) and the five malicious devices (Fig. 7(b)). We find that the trust values of honest devices increase to > 0.5 and are maintained at a high level. On the other hand, the trust values of malicious devices are all < 0.5. They will be filtered by the tracker during OLoc construction. Fig. 8 shows the number of total false validations by malicious devices.After a short duration, most malicious devices are not able to perform false validations.

Fig. 9 shows the performance of CCV varying with the percentage of malicious devices, ranging from 0% to 50%, after 1000s. At that time, most malicious devices will be detected and not used but the capacity of the whole cache pool decreases. Hence we find that the latency (Fig. 9(a)) and average number of descriptions (Fig. 9(b)) both increase with more malicious devices. The average message per certificate decreases from 1.88 to 1.77 in Fig. 9(c) because more individual validation is conducted. However, CCV protocol still achieves much better performance on average latency and number of local decryptions, compared to I-Valid.

False validation attack: The evaluation of trust value changes has been analyzed and the results, such as those in Fig. 8, indicate that after a short period of time, the malicious devices are not able to conduct false validation attacks.

Self-promoting attack: Trust value of each device is maintained and updated by the tracker. It is hard for a malicious device to promote itself to be a collaborator.

Defamation attack: Each collaborative validation event must carry a digital signature of the holder for authenticity and non-repudiation purposes. The digital signature will be verified by the tracker. Hence a malicious device cannot forge a false validation event from a honest device unless it owns the private key of the honest device.

Traitor attack and whitewashing attack: As shown in Fig. 7(b), once the malicious device provides bad devices, the trust value will drastically drop below the threshold, thus it can not be selected as a collaborator. Besides, the tracker can audit the identity of the device. Thus, the high cost will effectively prevent the whitewashing attack.

Collusion attack: Two or more malicious devices may improve their trust values by claiming that they helped each other. However, a malicious device will eventually provide a number of dishonest validations, which will be detected by the tracker statistically. In our model, the trust value reduction from one dishonest validation will be much larger than the trust value improvement from one honest validation. Hence it is only possible that the colluding malicious devices claim collaborative validations much more frequently than providing dishonest validations. Extremely high frequency of collaborative validations will also be detected by the tracker.

VI. Related Work

Certificate validation. Certificate-based PKIs are responsible for creating, managing, distributing, using, storing and revoking public key certificates, such as X.509. They are widely used in Web browsing (TLS), email (S/MIME) and document authentication. Efficient certificate validation has attracted a broad attention of the research community in recent years [11] [20] [2] [1] [16] [25] [12] [24]. Some approaches delegate validation task to a third party, such as a smart e-health gateway [20] and local ISPs [2]. Some approaches use prefetching and prevalidation techniques to reduce certificate validation cost [24], which can remove the time pressure from the certificate validation. However, this approach brings a huge cost for memory.

Trust Model. Ganeriwal et al. [9] proposes a distributed reputation-based framework (RFSN) for high integrity sensor networks. RFSN only uses direct trust between two nodes. Feng et al. [8] proposes the NBBTE algorithm to establish the direct trust and indirect values between two nodes by comprehensively considering and combining various factors. An efficient distributed trust model (EDTM) has been proposed in [13], which takes more trust metrics such as the energy level information into consideration besides communication behaviors. EDTM considers both direct and indirect trust.

VII. Conclusion

We design and evaluate the CCV protocol for fast public-key certificate validation. Our contributions include a memory-efficient and fast locator for certificate holders, called OLoc; a trust model for CCV to evaluate the trustworthiness of each device to avoid dishonest collaborative validation from malicious devices; and a complete protocol suite for efficient OLoc update, cache replacement, and revocation status checking mechanisms in a dynamic network. Evaluation results show that CCV significantly saves computation resource and certificate validation latency on IoT devices.

VIII. Acknowledgment

The authors are partially supported by National Science Foundation Grants CNS-1717948 and CNS-1750704.

References

[1] K. Akkaya, N. Saputro, S. Tonyali, M. Cebe, and M. Mahmoud. Efficient Certificate Verification for Vehicle-to-Grid Communications. Technical report, Florida Intl Univ., Miami, FL (United States), 2017.

[2] A. Alrawais, A. Alhothaily, J. Yu, C. Hu, and X. Cheng. Secureguard: a certificate validation system in public key infrastructure. *IEEE Transactions on Vehicular Technology*, 67(6):5399–5408, 2018.

[3] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 2002.

[5] D. Chen, G. Chang, D. Sun, J. Li, J. Jia, and X. Wang. TRM-IoT: A trust management model based on fuzzy reputation for internet of things. *Computer Science and Information Systems*, 8(4):1207–1228, 2011.

[6] K. Christidis and M. Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 2016.

[7] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[8] R. Feng, X. Xu, X. Zhou, and J. Wan. A trust evaluation algorithm for wireless sensor networks based on node behaviors and ds evidence theory. *Sensors*, 11(2):1345–1360, 2011.

[9] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(3):15, 2008.

[10] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[11] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle. Delegation-based Authentication and Authorization for the IP-based Internet of Things. In *Proc. of IEEE SECON*, 2014.

[12] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle. Towards viable certificate-based authentication for the internet of things. In *Proc. of ACM workshop on HotWiSec*, 2013.

[13] J. Jiang, G. Han, F. Wang, L. Shu, and M. Guizani. An efficient distributed trust model for wireless sensor networks. *IEEE transactions on parallel and distributed systems*, 26(5):1228–1237, 2015.

[14] A. Jøsang and T. Bhuiyan. Optimal trust network analysis with subjective logic. In *Proc. of IEEE SECURWARE*, 2008.

[15] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle. When HART goes wireless: Understanding and implementing the WirelessHART standard. In *Proc. of IEEE ETFA*, 2008.

[16] S. Kitajima and M. Mambo. Verifying the Validity of Public Key Certificates Using Edge Computing. In *Proc. of Springer SICBS*, 2017.

[17] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. In *Proc. of IEEE SP*, 2017.

[18] X. Li, H. Wang, Y. Yu, and C. Qian. An IoT Data Communication Framework for Authenticity and Integrity. In *Proc. of IEEE/ACM IoTDI*, 2017.

[19] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Proc. of ACM/IEEE IPSN*, 2013.

[20] S. R. Moosavi et al. Sea: a secure and efficient authentication and authorization architecture for iot-based healthcare using smart gateways. *Procedia Computer Science*, 52:452–459, 2015.

[21] R. Moskowitz and R. Hummen. Hip diet exchange (dex). *draft-moskowitz-hip-dex-00 (WiP), IETF*, 2012.

[22] A. Oracevic, S. Dilek, and S. Ozdemir. Security in internet of things: A survey. In *Proc. of IEEE ISNCC*, 2017.

[23] E. Rescorla and N. Modadugu. Datagram transport layer security version 1.2. 2012.

[24] E. Stark, L.-S. Huang, D. Israni, C. Jackson, and D. Boneh. The Case for Prefetching and Prevalidating TLS Server Certificates. In *Proc. of NDSS*, 2012.

[25] A. Wasef and X. Shen. EMAP: Expedite message authentication protocol for vehicular ad hoc networks. *IEEE transactions on Mobile Computing*, 12(1):78–89, 2013.

[26] Y. Yu, D. Belazzougui, C. Qian, and Q. Zhang. A concise forwarding information base for scalable and fast name lookups. In *Proc. of IEEE ICNP*, 2017.